An Experimental Analysis of DAG Scheduling Methods in Hard Real-time Multiprocessor Systems

[Extended Abstract] *

Manar Qamhieh
Université Paris-Est
LIGM
manar.qamhieh@univ-paris-est.fr

Serge Midonnet
Université Paris-Est
LIGM
serge.midonnet@univ-paris-est.fr

ABSTRACT

The scheduling of real-time parallel tasks on multiprocessor systems is more complicated than the one of independent sequential tasks, specially for the Directed Acyclic Graph (DAG) parallel model. The complexity is due to the structure of the DAG tasks and the precedence constraints between their subtasks. The trivial DAG scheduling method is to apply directly common real-time scheduling algorithms despite their lack of compatibility with the parallel model. Another scheduling method called the stretching method is summarized as converting each parallel DAG task in the set into a collection of independent sequential threads that are easier to be scheduled.

In this paper, we are interested in analyzing global preemptive scheduling of DAGs using both methods by showing that both methods are not comparable in the case of using Deadline Monotonic (DM) and Earliest Deadline First (EDF) scheduling algorithms. Then we use extensive simulations to compare and analyze their performance.

Keywords

Real-time scheduling, parallel tasks, Directed Acyclic Graphs, global preemptive scheduling, DM and EDF scheduling algorithms.

1. INTRODUCTION

Increasing the performance of execution platforms has been done by increasing the speed of uniprocessor systems associated to the reduction of the size of chips leading to heating problems. Multiprocessor systems have been seen as one solution to overcome these physical limitations, by increasing execution capabilities with processor parallelism. Many practical examples of shifting towards multiprocessors can be found, such as the *Intel Xeon*, *ARM* and *Cavium* processor families.

However, current real-time software APIs are not able to efficiently take advantage of multiprocessor platforms especially when parallelism is considered. In the industry, the majority of designed applications are targeting uniprocessor systems. But this is expected to change in the coming few years which will focus on parallel programming to take advantage of multiprocessor architectures. There are many models of parallel application, but in this work, we are interested in a particular family of parallelism called inter-subtask parallelism, in which a parallel task consists of a collection of subtasks under precedence constraints and dependencies between subtasks. The most general model is the Directed Acyclic Graph (DAG) model, which we consider as our task model in this paper.

Real-time scheduling of DAG tasks in particular and parallel tasks in general is a challenging problem. In hard real-time systems, the correctness of a system does not only depend on the correctness of the results, but on the respect of tasks timing parameters. Real-time systems on both uniprocessor and multiprocessor systems have been studied in the last decade, and many researches and scheduling algorithms have been proposed for such platforms [6]. However, the extension of real-time scheduling w.r.t. parallel tasks with dependencies is not trivial. Later in this paper, we present two main DAG scheduling methods which are used in literature. We call the first method the parallel scheduling method, in which parallel tasks are scheduled directly using common scheduling algorithms. The other method is the stretching scheduling method which transforms DAG tasks into independent sequential tasks that execute on multiprocessor systems. The stretching method simplifies scheduling at the price of losing some of the characteristics of parallel tasks, as it removes subtasks dependencies such that classical scheduling algorithms can then be used.

Both scheduling methods are used independently in many researches. To the best of our knowledge, this is the first work which compares both methods and show that both methods are not comparable regarding schedulability in the case of global preemptive Deadline Monotonic (DM) and Earliest Deadline First (EDF) scheduling algorithms. Then we compare their performance by extensive simulation and experimental results.

The remainder of this paper is organized as follows. Section 2 presents related works w.r.t. to the problem of scheduling real-time tasks on multiprocessor systems especially for parallel DAG model. Then we present in Section 3 the con-

*A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

RACS'14 October 5-8, 2014, Baltimore, MD, USA.

sidered task model. In Section 4, we describe in details the two DAG scheduling methods and we prove their incomparability. Simulation results are provided in Section 5 to compare the performance of these scheduling methods. Finally, Section 6 concludes this work.

2. RELATED WORK

The scheduling of parallel real-time tasks of different models has been studied on both uniprocessor and multiprocessor systems. In the case of uniprocessor systems, a classical approach in the state-of-the-art is to transform a parallel task into a chain and to assign each subtask of the chain extra timing parameters used for their scheduling. For example, in [4], the authors considered a hybrid taskset of periodic independent tasks and sporadic dependent graph tasks with a DAG model. They proposed an algorithm aiming at modifying the DAG timing parameters (by adding intermediate offsets and deadlines) in order to remove the dependencies between the tasks in the analysis.

In the case of multiprocessor systems, most research has been done regarding scheduling hard real-time tasks on homogeneous multiprocessor systems, especially for independent sequential tasks [6]. As mentioned above, there are mainly two methods for scheduling parallel DAGs in hard real-time systems. The parallel method schedules DAG tasks by using directly common scheduling algorithms and adapting the performance analysis and the scheduling conditions to take into consideration the particular characteristics of DAGs and parallel tasks in general. This technique is introduced in [1], which considers a taskset of a single sporadic DAG. They also provided polynomial and pseudo-polynomial schedulability tests for EDF scheduling algorithm.

The problem of scheduling multiple DAG tasks on multiprocessors have been more studied in [2, 9]. The authors considered general timing parameters of DAG tasks without taking into account their internal structure. In [10], the internal structure and the dependencies between subtasks are considered in the analysis of global EDF scheduling of DAG tasks.

The stretching method for DAG scheduling is based on DAG transformation. Dependencies of inter-subtask parallelism are removed and a DAG task is transformed into a collection of independent sequential threads. A decomposition algorithm is provided in [11] to distribute the slack time of the DAG task on its subtasks. The slack time is defined as the difference between the deadline of the graph and its minimum sequential execution time. The subtasks are assigned intermediate offsets and deadlines.

In [8], the authors considered fork-join model of parallel tasks and they proposed a stretching algorithm to execute them as sequentially as possible. A fork-join model is represented as an alternative sequence of sequential and parallel segment. All parallel segments of the same task have the same number of threads, and the threads of each segment have the same sequential execution length. The authors proposed to stretch a fork-join task into a master thread with utilization equal to 1, the remaining parallel threads are forced to execute in parallel with the master thread within a fixed activation interval. Hence, dependencies are no longer considered in the scheduling process.

3. TASK MODEL

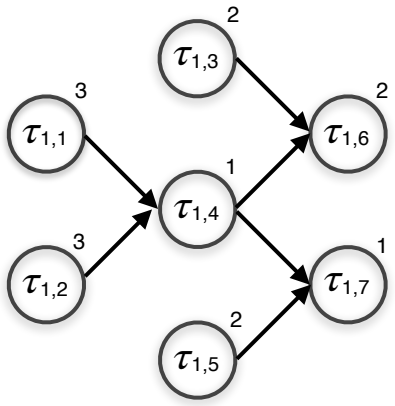


Figure 1: An example of a DAG task τ_1 which consists of 7 subtasks. The number on the upper right corner of each subtask represents its WCET and the arrows represent their precedence constraints.

We consider a taskset τ of n periodic parallel real-time Directed Acyclic Graph (DAG) tasks run on a system of m identical processors. The taskset τ is represented by $\{\tau_1, ..., \tau_n\}$. Each DAG task τ_i, where $1 \leq i \leq n$, is a periodic implicit-deadline graph which consists of a set of subtasks under precedence constraints that determine their execution flow. A DAG task τ_i is characterized by $(n_i, \{1 \leq j \leq n_i | \tau_{i,j}\}, G_i, D_i)$, where n_i is the number of its subtasks, the second parameter represents the set of subtasks of τ_i, G_i is the set of directed relations between these subtasks and D_i is τ_i's relative deadline. Since each DAG task has an implicit deadline, its period T_i (interval time between its successive jobs) is the same as its deadline $T_i = D_i$.

Let $\tau_{i,j}$ denotes the j^{th} subtask of the set of subtasks forming the DAG task τ_i, where $1 \leq j \leq n_i$. Each subtask $\tau_{i,j}$ is a single-threaded sequential task which is characterized by a WCET $C_{i,j}$. All subtasks of a DAG share the same deadline and period of the DAG. The total WCET C_i of DAG τ_i is defined as the sum of WCETs of its subtasks, where $C_i = \sum_{j=1}^{n_i} C_{i,j}$. Let U_i denote the utilization of τ_i where $U_i = \frac{C_i}{T_i}$.

The directed relations G_i between the subtasks of DAG τ_i define their dependencies. A directed relation between subtasks $\tau_{i,j}$ and $\tau_{i,k}$ means that $\tau_{i,j}$ is a predecessor of $\tau_{i,k}$, and the latter subtask have to wait for all of its predecessors to complete their execution before it can start its own. An example of a DAG task is shown in Figure 1, in which τ_1 consists of 7 subtasks. Precedence constraints are represented by directed arrows between subtasks. A source subtask is a subtask with no predecessors such as $\tau_{1,1}$. Respectively, a sink subtask is the one without any successors such as $\tau_{1,7}$.

Based on the structure of DAG tasks, the critical path of DAG τ_i is defined as the longest sequential execution path in the DAG when it executes on a virtual platform composed of an infinite number of processors. Its length L_i is the minimum response time of the DAG. A subtask that is part of the critical path is referred to as a critical subtask, while

non-critical subtasks are executed in parallel with the critical ones.

A DAG task is said to be feasible if the subtasks of all of its jobs respect its deadline. A taskset τ is deemed unfeasible when scheduled using any scheduling algorithm on m unit-speed processors if, at least, one of the following conditions is false:

$$\forall \tau_i \in \tau, \quad L_i \leq D_i$$
$$U(\tau) = \sum_{i=1}^{n} U_i \leq \sum_{i=1}^{n} \frac{C_i}{T_i} \leq m$$

4. DAG SCHEDULING METHODS: PARALLEL VS. STRETCHING

A real-time scheduling algorithm is responsible for distributing the executing jobs of the system on its available processors during the execution interval of the task set. At each time unit within this interval, the scheduling algorithm chooses particular jobs among the ready ones to execute on the system's processors using a particular priority assignment. A **global** real-time scheduler allows migration of any job in the system between the processors of the system during its execution, and a **preemptive** scheduler allows higher priority jobs to interrupt the execution of those of lower priority.

In this work, we consider two common real-time scheduling algorithms. The first is the Earliest Deadline First (EDF) algorithm which is an optimal [1] scheduling algorithm on uniprocessor systems and it performs well on multiprocessor systems despite the loss of its optimality. This algorithm assigns priorities to jobs based on their absolute deadlines where jobs with earlier absolute deadlines have higher priorities. Since jobs of the same real-time task might have different priorities, EDF is classified as a fixed job priority assignment algorithm. The second scheduling algorithm is the Deadline Monotonic (DM) algorithm from the fixed task priority assignment algorithms. In this algorithm, priorities assigned to tasks based on their relative deadline. Hence, all jobs of the same task have the same priority.

In the case of independent sequential real-time tasks, the scheduling algorithm can take decisions based on the timing parameters of tasks such as their deadline (absolute or relative), their period, their slack time, ... While, in parallel real-time tasks such as the DAG model, a single task consists of a collection of subtasks, each with a different activation time based on the structure of the DAG. According to the task model presented in Section 3, the timing parameters are assigned on a DAG level and not a subtask level, and subtasks are characterized by their worst-case execution time and their precedence relations only. The other timing parameters of subtasks necessary for the scheduling process are inherited from their original DAGs. Regular scheduling algorithms which are designed originally for the scheduling of sequential independent tasks can be used for the scheduling of such parallel tasks. However, it is harder for them to take accurate decisions for subtasks based on these given timing parameters.

We define a scheduling method of a parallel DAG task as the method which specifies the structure of subtasks in the DAG and how they execute w.r.t. their successor and predecessor subtasks. A scheduling method is used combined with a scheduling algorithm in order to execute parallel DAGs on multiprocessor systems. There are two scheduling methods for DAG tasks on multiprocessor systems which uses common scheduling algorithms in real-time systems, which we call **parallel** and **stretching** scheduling methods. More details and examples regarding the two scheduling methods of DAGs are provided below.

[1]Optimal scheduling algorithm is defined in Definition 1.

4.1 Parallel Scheduling Method

We consider the parallel scheduling method as the first and default scheduling method for DAG tasks on multiprocessor systems. This method supports inter-subtask parallelism in DAGs in which all subtasks are assumed to execute in parallel whenever it is possible. For example, if there are two sibling subtasks (who share the same parent) and there is more than one available processor at their activation time, then both subtasks will execute in parallel, otherwise, the scheduling algorithm blocks the execution of one of them and they will execute sequentially as a result.

According to the parallel scheduling method, the subtasks of a DAG task execute as soon as possible based on the default structure of the DAG task. For each subtask, we consider its earliest activation time which is the result of execution blocking of its predecessors while executing on a system of infinite number of processors. This method is used in literature and many analyses are provided, such as in [1, 2, 9, 10].

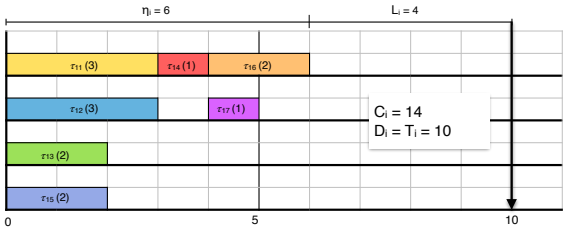


Figure 2: An example of parallel scheduling method of DAG task from Figure 1

We believe that the parallel scheduling method is better described using an example. Figure 2 shows the structure of subtasks of DAG task τ_1 from Figure 1 when the parallel scheduling method is applied. We assume that DAG task τ_1 has a deadline equal to 10 and it consists of 7 subtasks. As shown in Figure 1, the source subtasks $\{\tau_{1,1}, \tau_{1,2}, \tau_{1,3}, \tau_{1,5}\}$ are activated at time $t = 0$ and they are assumed to execute in parallel. Subtask $\tau_{1,4}$ is a successor of subtasks $\tau_{1,1}$ & $\tau_{1,2}$ and it has to wait at for them both to finish execution before it starts its own. In this example, the earliest possible of $\tau_{1,4}$ is equal to 3. By following the same reasoning, we calculate the earliest activation and even the latest finish time of subtasks based on their precedence constraints. Based on these additional timing parameters, we define the maximum possible execution interval of each subtask in a DAG task, in which it executes either in parallel or sequentially with other subtasks in the system based on the decisions of the scheduling algorithm.

4.2 Stretching Scheduling Method

The problem of dependencies between subtasks of a single DAG is solved using the stretching scheduling method.

In this method, a DAG task is converted into independent sequential threads, one of them, at least, is fully stretched. According to the stretching method, the parallel structure of DAGs is avoided whenever it is possible, and some of the subtasks are forced to execute sequentially, while the rest are assigned intermediate offsets and deadlines so as to execute independently. The DAG stretching technique is inspired from the algorithm provided in [8] which targets a special parallel task model called the Fork-join model. Briefly, the critical path of a DAG task is filled by some of the non-critical subtasks until it is fully stretched up to its deadline. Based on this stretching, the remaining subtasks of the DAG are forced to execute in parallel with the stretched critical path. In order to maintain the original precedence constraints in the DAG, the subtasks are assigned intermediate offsets and deadlines which assure their independence. As a result, the scheduling problem is simplified into the common scheduling problem of independent sequential threads on multiprocessor systems.

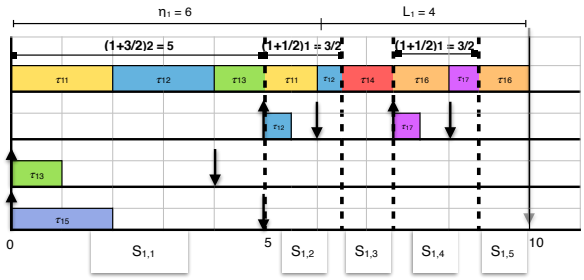


Figure 3: Example of stretching scheduling method for DAGs.

Figure 3 shows the structure of the DAG task from Figure 1 when the stretching scheduling method is applied. Let $S_{i,j}$ denote the j^{th} segment of task τ_i. As shown in the figure, the critical path is transformed into a sequential thread of WCET equal to 10 which is the deadline of the DAG, while the remaining sibling subtasks execute in parallel. For more details about the stretching algorithm for parallel tasks, we invite the reader to refer to [8].

4.3 In-comparability of Scheduling Methods

In this section, we compare the parallel and stretching methods of DAG scheduling. We discuss the case of global preemptive scheduling in which two scheduling algorithms are used. The algorithms are the Deadline Monotonic (DM) from the fixed task priority assignment family and the Earliest Deadline First (EDF) from the fixed job family.

Definition 1. *A scheduling algorithm is said to be* ***optimal*** *if it is able to schedule all possible feasible task sets.*

Definition 2. *Scheduling algorithm $\mathcal{A}$ dominates scheduling algorithm $\mathcal{B}$ if all task sets schedulable by algorithm $\mathcal{B}$ are also schedulable by algorithm $\mathcal{A}$, but the opposite is not correct.*

Using two examples, we show that both DAG scheduling methods presented in this paper are not optimal. This is not considered as a drawback of these approaches because in real-time multiprocessor scheduling, optimal scheduling algorithms are not practical due to huge overheads and execution costs of algorithms. Also, common uniprocessor scheduling algorithms cease to be optimal when they are used on multiprocessors. As a result, non-optimal scheduling algorithms are used for multiprocessor scheduling in practice.

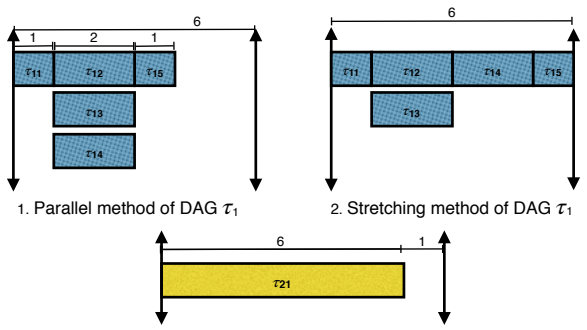
Proving the in-comparability of both scheduling methods is important in order to show that both methods are acceptable for DAG scheduling and no one dominates the other from schedulability point of view. In order to do so, we provide two general examples showing the scheduling of a given DAG set on multiprocessor system using a global preemptive scheduling algorithm. In the first example, we show that the DAG set is schedulable when the stretching scheduling method is used, while the parallel scheduling method leads to a deadline miss. In the second example, we show oppositely that parallel scheduling method schedules successfully a DAG set while stretching method fails. The following examples are valid for global EDF and DM scheduling algorithms.

Usually, finding a single case of in-comparability is enough to deem the performance of a scheduling method or algorithm.

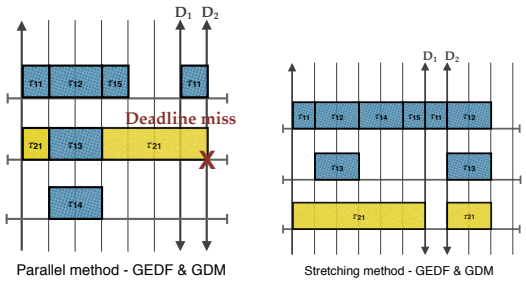
Example 1

Taskset: As shown in Figure 4, the DAG set of this example consists of two periodic implicit-deadline DAG tasks. DAG τ_1 has a deadline equal to 6, and it consists of 5 subtasks. The worst-case execution time of each subtask and the precedence constraints between them are shown in Inset 1 of Figure 4(a). This inset represents also the structure of DAG τ_1 when parallel scheduling method is used, which favored inter-subtask parallelism. Hence, subtasks of DAG τ_1 execute as soon as possible by considering an infinite number of processors. Subtask $\tau_{1,1}$ is a source subtask which is activated by the activation of the DAG. Then, subtasks $\{\tau_{1,2}, \tau_{1,3}, \tau_{1,4}\}$ execute in parallel. When they finish execution, the sink subtask $\tau_{1,5}$ executes. Inset 2 of Figure 4(a) shows the structure of DAG τ_1 in the case of stretching scheduling method, in which its critical path $(\{\tau_{1,1}, \tau_{1,2}, \tau_{1,5}\})$ is filled with non-critical subtasks until its deadline. The result is a fully stretched thread (its worst-case execution time is equal to the deadline) and subtask $\tau_{1,3}$ which executes in parallel with it. Task τ_2 is shown in Inset 3 of Figure 4(a) and it has a deadline equal to 7. It consists of a single subtask $\tau_{2,1}$ which has a worst-case execution time equal to 6. Since task τ_2 is a sequential task, there is no difference between its parallel and stretching scheduling methods and the result is the same as shown in the inset.

Parallel method: Figure 4(b) shows the global preemptive scheduling of DAG set on a system consists of 3 homogeneous processors. This scheduling is done based on the parallel scheduling method of DAGs and for both scheduling algorithms, EDF and DM. According to DM algorithm, all jobs of DAG τ_1 have higher priority than the jobs of DAG τ_2. Also in the case of EDF, the first job of DAG τ_1 has an earlier absolute deadline than the first job of DAG τ_2 when synchronous activation of DAGs is considered. As shown in the figure, the first job of τ_1 executes without being interrupted by the other task, and since we consider a parallel scheduling method, its parallel subtasks $(\{\tau_{1,2}, \tau_{1,3}, \tau_{1,4}\})$ occupy the processors of the system for 2 time units. As a result, the first job of τ_2 is blocked during this time. Know-



(a) Example 1: DAG set consists of two periodic implicit-deadline DAG tasks. (1) The structure of DAG τ_1 when the parallel scheduling method is used. (2) The structure of DAG τ_1 when the stretching scheduling method is used. (3) Both scheduling methods (parallel & stretching) of DAG τ_2.



(b) Global EDF & DM scheduling deadline miss when parallel method for DAG set.

(c) A successful global EDF & DM scheduling in the case of stretching method is used for DAG set.

Figure 4: An example of scheduling method incomparability in favor of the stretching method.

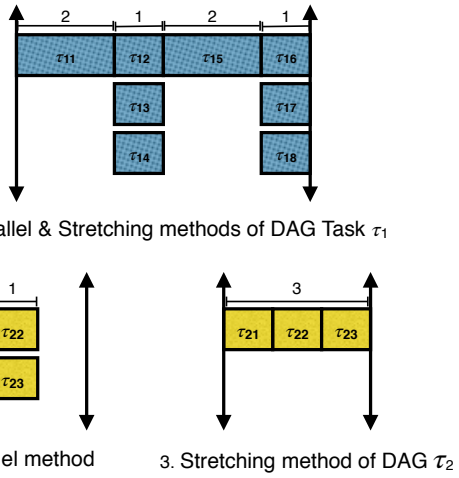
ing that DAG task τ_2 has 1 time unit as a slack, a definite deadline miss occurs due to this blocking.

Stretching method: The same characteristics of the above scheduling are considered for the case of stretching scheduling method. Based on the structure of DAG task τ_1 in Inset 2 of Figure 4(a), its jobs need a maximum of 2 processors in order to execute successfully at all times, because the stretching method forces subtask $\tau_{1,4}$ to execute sequentially within the critical path of the DAG. For any given scheduling algorithm, the DAG set is schedulable on a system of 3 processors, since DAG τ_1 needs only 2 processors to execute and the remaining processor can be used by DAG task τ_2.

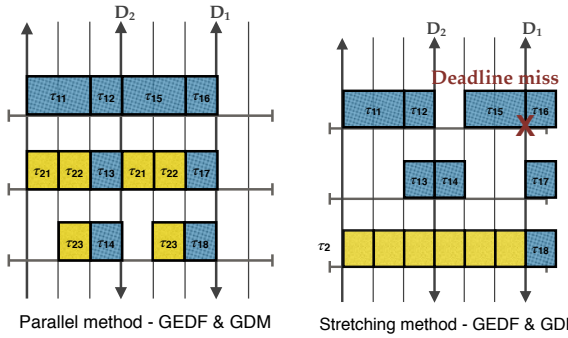
Conclusion: In the case of global preemptive EDF and DM scheduling algorithms, stretching scheduling algorithm succeeds in scheduling the periodic implicit-deadline DAG set while the parallel method fails.

Example 2

Taskset: As shown in Figure 5, the DAG set of this example consists of two periodic implicit-deadline DAG tasks. DAG τ_1 has a deadline equal to 6, and it consists of 4 segments of 8 subtasks. The worst-case execution time of each subtask and the precedence constraints between them are



(a) Example 2: DAG set consists of two periodic implicit-deadline DAG tasks. (1)Both scheduling methods (parallel & stretching) of DAG τ_1. (2) The structure of DAG τ_2 when the parallel scheduling method is used. (3) The structure of DAG τ_2 when the stretching scheduling method is used.



(b) A successful global EDF & DM scheduling in the case of parallel method for DAG set.

(c) Global EDF & DM scheduling showing a deadline miss when stretching method is used for DAG set.

Figure 5: A scheduling example of a DAG set scheduled using parallel scheduling.

shown in Inset 1 of Figure 5(a). Since DAG τ_1 has no slack time (its critical path length is equal to its deadline), applying the stretching scheduling method does not change the structure of the DAG from the parallel method, hence, Inset 1 of Figure 5(a) represents both scheduling methods. DAG task τ_2 consists of 3 subtasks, in which subtask $\tau_{2,1}$ is the source subtask of the DAG and subtasks $\{\tau_{2,2}, \tau_{2,3}\}$ are its successors. According to the parallel scheduling method, both subtasks $\tau_{2,2}$ and $\tau_{2,3}$ execute in parallel as shown in Inset 2 of Figure 5(a). We consider the critical path of the DAG τ_2 to be subtasks $\{\tau_{2,1}, \tau_{2,2}\}$ and its slack is equal to 1 time unit. Using the stretching scheduling method, all subtasks are forced to execute sequentially as shown in Inset 3 of Figure 5(a). In this method, subtask $\tau_{2,3}$ is forced to execute after subtask $\tau_{2,2}$ and all subtasks of DAG τ_2 form a single sequential fully stretched thread.

Parallel method: In Figure 5(b), we show the global preemptive scheduling of DAG set on a system of 3 iden-

tical processors using the parallel scheduling method. This example can be applied to EDF and DM scheduling algorithms as in Example 1. Based on DM, all jobs of DAG τ_2 have higher priorities than the ones of DAG τ_1, because relative deadline of τ_2 is lower than the relative deadline of τ_1 ($3 < 6$). In the case of EDF, which assigns priorities to jobs based on their absolute deadlines, the first job of DAG τ_2 has the earliest absolute deadline, then it has the highest priority. However, the first job of DAG τ_1 and the second job of DAG τ_2 have the same absolute deadline, then we consider that the second job of τ_2 has higher priority arbitrary. According to the parallel scheduling method, subtasks $\tau_{2,2}$ & $\tau_{2,3}$ can execute in parallel whenever there are processors available in the system. As shown in Figure 5(b), all subtasks of DAG τ_2 execute in time interval [0,2) and [3,5) and they do not interrupt the execution of DAG τ_1 despite the priority ordering. According to this scheduling, no deadline miss occurs.

Stretching method: According to the stretching scheduling method, subtasks of DAG τ_2 are forced to execute sequentially as a single thread. Since it has higher priority according to DM and EDF, DAG task τ_2 occupies a single processor by itself, which leads to a deadline miss for the first job of DAG τ_1. This is a result of the blocking effect and the delay of subtask $\tau_{1,4}$ in time interval [2,3).

Conclusion: In the case of global preemptive EDF and DM scheduling algorithms, parallel scheduling algorithm succeeds in scheduling the periodic implicit-deadline DAG set while the stretching method fails. What we conclude from both examples is that both scheduling methods are not comparable and no one dominates the other.

5. SIMULATION ANALYSIS

Due to the in-comparability of the parallel and stretching scheduling methods in the case of global preemptive DM and EDF on multiprocessor systems proved by the above examples, we use extensive simulation as an indication to the performance of both methods. Simulation is a good tool to give us indications about the performance of particular scheduling methods and algorithms and it is useful when we compare scheduling algorithms together.

Simulation Environment

The simulation process is based on an event-triggered scheduling. This means that at each event in the interval $[0, H)$, where H denotes the hyper period of the scheduled taskset τ and defined as the least common multiple of periods, the scheduler is awakened and it decides which jobs have the highest priorities to execute on the available processors.

We used a simulation tool called YARTISS [3], which is a multiprocessor real-time scheduling simulator developed in Java. It contains many scheduling algorithms and task models (including parallel tasks), and it can be used easily for both hard and soft real-time systems.

For each system utilization from 1 to 8, we generated $50,000$ tasksets randomly. The number of parallel tasks within each taskset is varied from 2 to 10 tasks/taskset. This variation affects the structure of tasks because, for a fixed utilization, increasing the number of tasks will decrease their WCET in average and as a result, the number of parallel threads per task will be lowered. Based on this, we can control the percentage of parallelism within a taskset and it can help in analyzing the effect of parallelism on scheduling as we will see below.

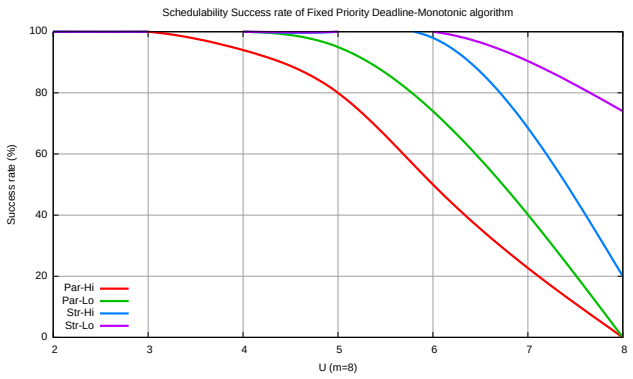


Figure 6: Simulation results of global DAG fixed priority deadline monotonic scheduling.

Regarding the generation of parallel tasksets, our task generator is based on the Uunifast-Discard algorithm [5] for random generation of tasks. This algorithm is proposed by Davis and Burns to generate randomly a set of tasks of a certain total utilization on multiprocessor systems. The number of tasks and their utilization are inputs of this algorithm. The taskset generator is described briefly as follows:

- The algorithm takes two parameters n and U, where n is the number of parallel tasks in the set and U is the total utilization of the taskset ($U > 0$).
- The Uunifast-Discard algorithm distributes the total utilization on the taskset. A parallel task τ_i can have a utilization U_i greater than 1 which means that its threads cannot be stretched completely, and it has to execute in parallel.
- The number of threads and their WCET of each parallel tasks are generated randomly based on the utilization of the tasks. The maximum number of threads is predefined as 10 threads per parallel task.
- The number of parallel tasks per taskset is varied as well in the generation so as to vary the average utilization per task. Tasksets with low number of tasks tend to have high utilization per task (Hi), while tasksets with high number of tasks have lower task's utilization (Lo).

In order to limit the simulation interval and reduce the time needed to perform the simulation, which is based on the length of the hyper period of each taskset, we used the limitation method proposed in [7], which relays on using a considerate choice of periods of the tasks while generation so as to reduce their least common multiple. Using this method, we implemented our task generator to choose periods of tasks in the interval $[1, 25200]$.

Simulation Results

The performance of both scheduling methods varies significantly based on the used scheduling algorithm. This result is not clear using the scheduling analysis and it is possible to show it using simulation only.

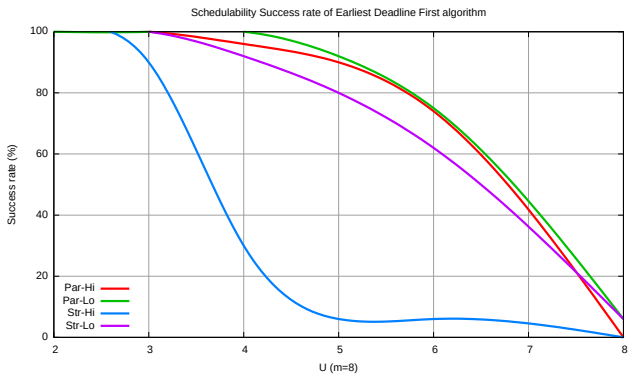


Figure 7: Simulation results of global DAG EDF scheduling.

We compare the scheduling success rate of simulated tasksets while varying their utilization on a fixed number of processors in the system. Starting by Figure 6, we can notice that, in the case of global preemptive fixed task priority DM algorithm, the stretching scheduling method (in both taskset types, Hi and Lo) has better success rates than the parallel scheduling method. And almost 100% of task sets remained schedulable using stretching method while the schedulability of task sets using parallel scheduling started to decrease from utilization equal to 3. We can conclude that the stretching method is more adapted to fixed task priority assignment algorithm.

While in Figure 7, parallel scheduling method performs better in the case of global preemptive EDF scheduling algorithm, which means that this method is more adapted to the fixed job priority assignment scheduling algorithms. We can notice that the schedulability of stretched task sets with high utilization (str-hi) decreases sharply little before system's utilization become 3 to reach less than 10% of success rate, while the parallel scheduling method performs well in general.

Based on these results, we conclude that even if both scheduling methods are not comparable using scheduling analysis and there is no definite dominance of one on another, it is still interesting to use a more adapted scheduling method based on the chosen scheduling algorithm.

6. CONCLUSION

In this paper, we described two main scheduling methods for global preemptive parallel real-time DAG tasks on multiprocessor systems. The parallel scheduling method enforces inter-subtask parallelism and allows direct scheduling for parallel tasks. While stretching scheduling method transforms parallel tasks into sequential independent task model easier to be scheduled. Both scheduling methods have advantages and disadvantages, and we prove, using scheduling examples, that they are not comparable regarding scheduling and no one of them dominates the other. The comparability analysis is shown for two common scheduling algorithms, the Deadline Monotonic (DM) from the fixed task priority assignment family and the Earliest Deadline First (EDF) from the fixed job priority assignment family.

We conclude the paper by simulation results that show clearly that DM algorithm is more adapted to the stretching scheduling method while EDF performs better when DAG tasks execute using parallel method. In the future, we aim at better analyzing the behavior of such algorithms on the scheduling methods of DAG scheduling, and we aim at extending the analysis to consider other scheduling algorithms may be found in literature.

7. REFERENCES

[1] S. K. Baruah, V. Bonifaciy, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A Generalized Parallel Task Model for Recurrent Real-time Processes. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS)*, pages 63–72. IEEE Computer Society, Dec. 2012.

[2] V. Bonifaci, A. Marchetti-spaccamela, S. Stiller, and A. Wiese. Feasibility Analysis in the Sporadic DAG Task Model. In *25th euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[3] Y. Chandarli, F. Fauberteau, M. Damien, S. Midonnet, and M. Qamhieh. YARTISS: A Tool to Visualize, Test, Compare and Evaluate Real-Time Scheduling Algorithms. In *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2012.

[4] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[5] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[6] R. I. Davis and A. Burns. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. *ACM Computing surveys*, pages 1 – 44, 2011.

[7] J. Goossens and C. Macq. Limitation of the Hyper-Period in Real-Time Periodic Task Set Generation. In *Proceedings of the 9th International Conference on Real-Time Systems (RTS)*, pages 133–148, Mar. 2001.

[8] K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar. Scheduling Parallel Real-Time Tasks on Multi-core Processors. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS)*, pages 259–268. IEEE Computer Society, 2010.

[9] A. J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of Global EDF for Parallel Tasks. In *Euromicro Conference on Real-Time Systems ECRTS*, number 314, 2013.

[10] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet. Global edf scheduling of directed acyclic graphs on multiprocessor systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, RTNS '13, pages 287–296, New York, NY, USA, 2013. ACM.

[11] A. Saifullah, D. Ferry, K. Agrawal, C. Lu, and C. Gill. Parallel real-time scheduling of DAGs. In *IEEE Transactions on Parallel and Distributed Systems*, 2014. accepted.